



REVERSING CON IDA PRO

Gerardo Fernández <gerardofn@virustotal.com>

¿Reversing?

- Llamamos al reversing a todo proceso de extracción de información sobre el funcionamiento interno de una aplicación (o dispositivo) sin conocimiento de cómo está construido → Ingeniería Inversa.

¿IDA Pro?

- IDA Pro es una aplicación multiplataforma que permite desensamblar, depurar y decompilar ficheros binarios ejecutables.

Ejemplo

```
int main(int argc, char* argv[ ]) {
    if (argc != 3) return EXIT_FAILURE;
    if (strncmp(argv[1], "-f", 2) == 0) infectarFichero(argv[2]);
    return EXIT_SUCCESS;
}
```

Desensamblado

```
cmp    [ebp+argc], 3          ; if (argc != 3) return EXIT_FAILURE;
jz     short loc_401020
mov    eax, 1
jmp    short loc_40105B          ; if (argc != 3) return EXIT FAILURE;
```

loc_401020:

Procesamiento de parámetros {

```
push   2
push   offset Str2
mov    eax, 4
shl    eax, 0
mov    ecx, [ebp+argv]
mov    edx, [ecx+eax]
push   edx
call  ds:strncmp
; if (strcmp(argv[1],"-f",2)==0) infectarFichero(argv[2]);
; if (strcmp(argv[1],"-f",2)==0) infectarFichero(argv[2]);
```

Procesamiento del resultado {

```
add    esp, 0Ch
test   eax, eax
jnz   short loc_401059
mov    eax, 4
shl    eax, 1
mov    ecx, [ebp+argv]
mov    edx, [ecx+eax]
push   edx
call  sub_401000
; if (strcmp(argv[1],"-f",2)==0) infectarFichero(argv[2]);
; if (strcmp(argv[1],"-f",2)==0) infectarFichero(argv[2]);
```

```
add    esp, 4
; EXIT_FAILURE
```

loc_401059:

```
xor    eax, eax
; EXIT_SUCCESS
```

loc_40105B:

```
pop    ebp
ret
```

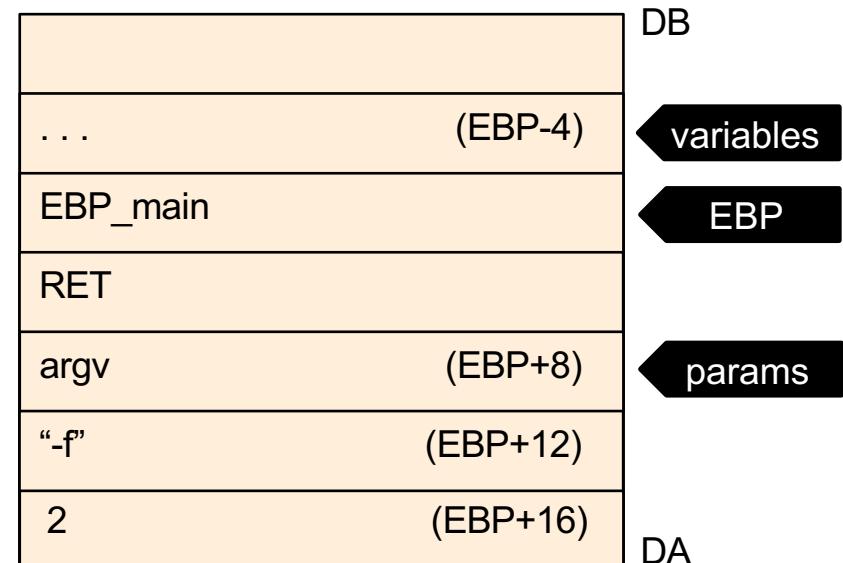
Desensamblado

```
cmp    [ebp+argc], 3
jz     short loc_401020
mov    eax, 1
jmp    short loc_40105B

loc_401020:
        push    2
        push    offset Str2
        mov     eax, 4
        shl     eax, 0
        mov     ecx, [ebp+argv]
        mov     edx, [ecx+eax]
        push   edx
        call   ds:strncmp ← ----- -
        add    esp, 0Ch
        test   eax, eax
        jnz    short loc_401059
        mov    eax, 4
        shl    eax, 1
        mov    ecx, [ebp+argv]
        mov    edx, [ecx+eax]
        push   edx
        call   sub_401000
        add    esp, 4

loc_401059:
        xor    eax, eax
loc_40105B:
        pop    ebp
        retn
```

Pila



Visión de la pila dentro de la llamada a **strcmp**

¿Cómo nos enfrentamos a esto?

APROXIMACIÓN

- **Objetivo:** realizar el análisis en el menor tiempo posible.
 - Localizar el punto de entrada (real).
 - Buscar **disparadores** del análisis:
 - Cadenas significativas
 - Funciones API relevantes
 - Nombres de segmentos sospechosos
 - Calls inusuales
 - Funciones muy referenciadas
 - Valores de retorno de las funciones
 - Malware: consultar BDs de inteligencia, obtener código comunes, etc.

HERRAMIENTAS

- **Referencias cruzadas**
- **Trazado de la ejecución**
- **Breakpoints** de distintos tipos: [in]condicionales, en memoria, en librerías, software o hardware, etc.
- Representación en **grafo**
- Manipulando la **interpretación** del desensamblado: etiquetado, renombrado, resaltado, agrupando bloques de código, etc.
- Modificando el flujo de **ejecución** para estudiar respuesta o disparar una acción.
- Utilización de lenguajes de **programación** para automatizar tareas.

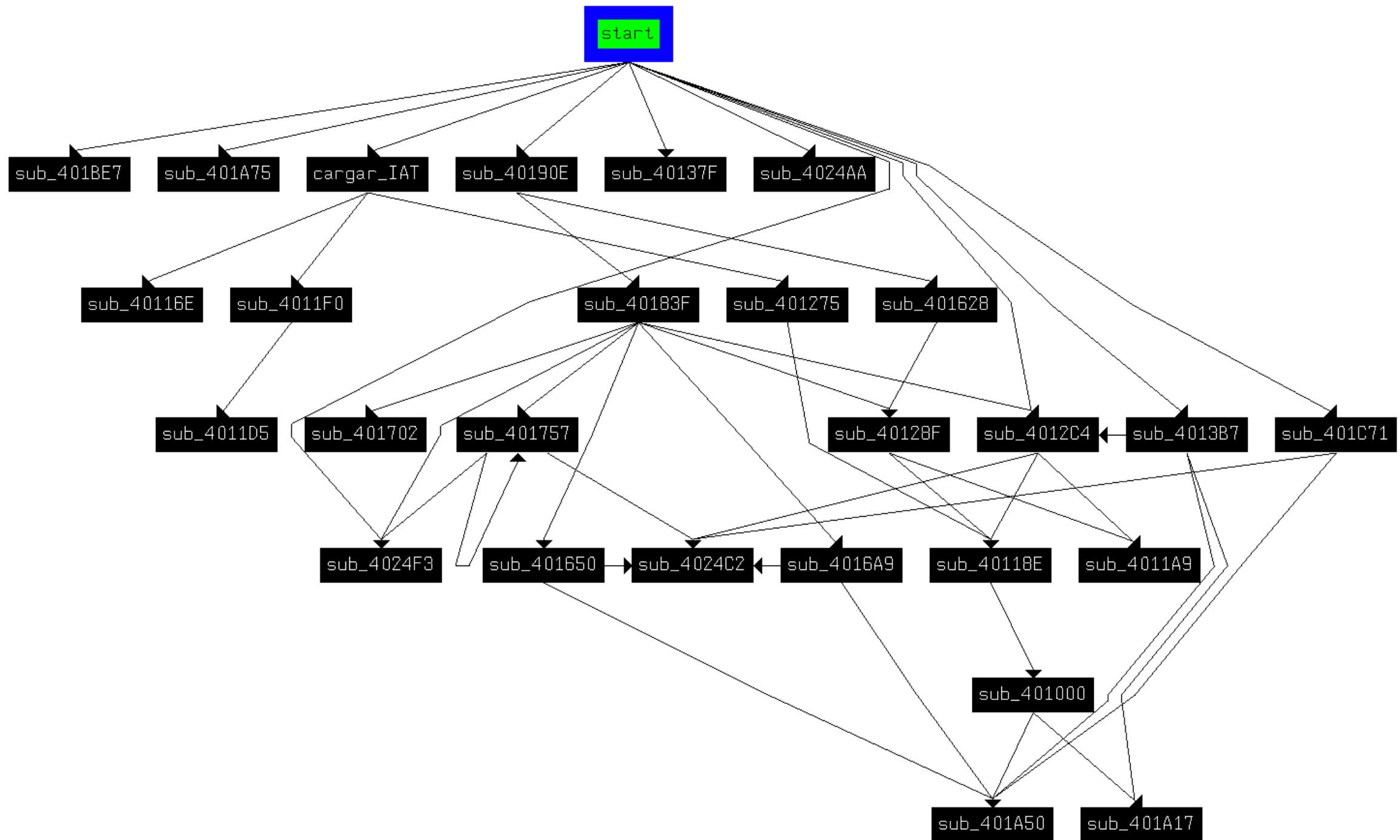
TÍTULO DE EXPERTO UNIVERSITARIO EN

INGENIERÍA INVERSA E INTELIGENCIA MALWARE



El reto de “Reconstruir IAT y Xrefs”

- Tras extraer la carga oculta en una muestra de malware nos encontramos que..
 - No tiene tabla de importaciones
 - Por lo que no hay referencias a llamadas a la API de Windows.
- El alumno debe crear una rutina en **IDAPython** para reestablecer las referencias cruzadas.



```
__author__ = 'Oscar Martín Vicente'
```

```
JMPS = [idaapi.NN_jmp, idaapi.NN_jmpfi, idaapi.NN_jmpni]  
CALLS = [idaapi.NN_call, idaapi.NN_callfi, idaapi.NN_callni]
```

```
for func in idautils.Functions():  
    flags = idc.get_func_attr(func, FUNCATTR_FLAGS)  
    if flags & FUNC_LIB or flags & FUNC_THUNK:  
        continue  
    dism_addr = list(idautils.FuncItems(func))
```

} Para cada función (completa) de usuario

```
for ea in dism_addr:  
    idaapi.decode_insn(ea)
```

```
    if idaapi.cmd.type in JMPS or idaapi.cmd.type in CALLS:  
        if idc.get_operand_type(ea, 0) == 2:  
            iat_offset_addr = idc.get_operand_value(ea, 0)  
            idc.AddCodeXref(ea, iat_offset_addr, idc.XREF_USER | idc.fl_CN)
```

CALL
o
JMP

```
    elif idc.get_operand_type(ea, 0) == 1:  
        disasm = idc.generate_disasm_line(ea, 0)  
        parts = disasm.split(";")  
        if len(parts) == 2:  
            funcName = parts[1].strip()  
            iat_offset_addr = idc.get_name_ea_simple(funcName)  
            if iat_offset_addr != idc.BADADDR:  
                idc.AddCodeXref(ea, iat_offset_addr, idc.XREF_USER | idc.fl_CN)
```

Operando es una
referencia directa a
memoria

Operando es un
registro

El reto de la "Carga oculta"

- El objetivo es encontrar la carga oculta y analizarla.
- El alumno debe lidiar con:
 - Funciones que no hacen nada.
 - Detección de la presencia de breakpoints.
 - Llamadas a la API de Windows que no tienen sentido.
 - Llamadas a la API de Windows que, en realidad, no se efectúan.
 - Llamadas a la API ofuscadas.
 - Algoritmo de cifrado “custom”.



REVERSING CON IDA PRO

Gerardo Fernández <gerardofn@virustotal.com>